Sujet de TP 8 Réseaux - sockets L3 Info-M1 ÉLISE/UFR ST/UMLP

Programmation réseau avec sockets en C sous Linux (3)
Échange d'une donnée de grande taille - protocole TCP
Des centaines de milliers de décimales du nombre π

V. Felea & A. Hugeat & E. Merlet

Dans ce TP, une application souhaite informer une autre application de la valeur du nombre π avec une très bonne précision. Le serveur envoie la valeur du nombre π au client. La valeur du nombre π à transmettre a 100000 décimales (voir le fichier donné sous Moodle pour les premières 100000 décimales du nombre π). Vous allez devoir écrire les deux applications communicantes où cette valeur est lue depuis le fichier et est transmise sous la forme d'une chaîne de caractères.

Indication. La lecture de la valeur π depuis le fichier peut être réalisée par les fonctions `fopen` (ouverture de fichier), `fscanf` (lecture depuis un fichier).

1. Transmission d'une "faible" quantité d'octets

Q1 Tester l'application avec l'envoi des 102 premiers caractères lus dans le fichier (le chiffre 3, le point et les premières 100 décimales du nombre π).

Q2 Modifier l'application afin que seules les premières 100 décimales soient envoyées (le chiffre 3 et le point ne doivent plus être envoyés).

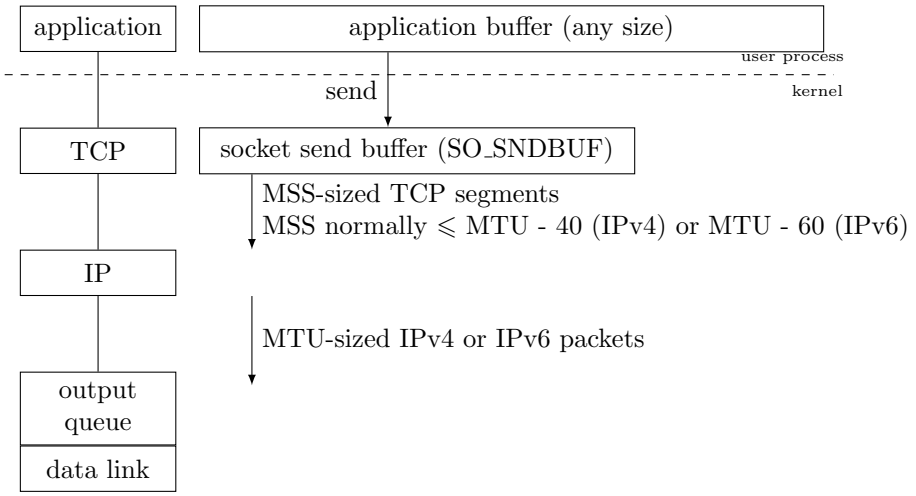
Q3 Considérer que l'application réceptrice peut recevoir uniquement 10 décimales à la fois. Modifier le programme du client en conséquence (**sans modification du serveur**) et afficher les réceptions intermédiaires. À la fin des réceptions, les 100 décimales reçues doivent être placées dans une seule chaîne de caractères.

Q4 Quelles conclusions tirez-vous de la question précédente ?

2. Tampons internes associés aux sockets

Pour chaque connexion TCP établie, le protocole TCP envoie et reçoit les segments en passant par des tampons internes - appelés aussi buffers (un pour les transmissions, un pour les réceptions) gérés par le noyau.

L'exploitation de la pile protocolaire TCP/IP lors de l'émission d'une donnée (issue de l'applicatif, par la fonction `send` des sockets) est illustrée dans la figure suivante.



Légende :

- MSS (Maximum Segment Size) = taille maximale de la donnée utile (i.e. donnée applicative) d'un segment
- MTU (Maximum Transmission Unit) = taille de la donnée utile (i.e. donnée applicative + entêtes TCP et IP) d'une trame

Les données à envoyer sont copiées dans le tampon interne associé à la socket. Le protocole de transport (ici TCP) crée autant de segments que nécessaire, chacun contenant un sous-ensemble de données à envoyer. Ces segments sont potentiellement encore divisés (en fragments), dans la couche Internet/Réseau, par le protocole IP.
Chez le récepteur, les fragments issus d'un même segment sont assemblés et les données des segments sont placées dans le tampon interne de la socket réceptrice. Les appels de la fonction `recv` consommeront les données depuis ce tampon interne pour les déposer dans la mémoire du processus, spécifiée par l'adresse donnée en deuxième paramètre de la fonction.

Indication concernant la visualisation du comportement de la communication TCP Pour obtenir des informations sur le nombre d'octets présents dans les tampons internes, la commande Linux `ss` fournit :

- dans la colonne **Recv-Q** : le nombre d'octets présents dans le tampon interne de réception. Il s'agit d'octets reçus chez le récepteur, mais non encore lus par l'applicatif.
- dans la colonne **Send-Q** : le nombre d'octets présents dans le tampon interne d'émission. Ce tampon est alimenté par l'application émettrice. Il contient des octets non acquittés par le récepteur.

Remarque. Ces informations sont aussi disponibles pour les échanges en mode non connecté (par le protocole de transport UDP). Pour chaque type de communication, les sockets correspondantes (en mode connecté pour TCP et en mode non connecté pour UDP) devraient être sélectionnées.

Objectifs de la suite du TP : comprendre la notion de buffers internes associés à la socket, ainsi que les mécanismes de transfert entre les buffers internes manipulés par le protocole TCP.

2.1. Transmission d'une "importante" quantité d'octets

Q1 Modifier les applications pour transmettre les 100002 caractères de π depuis le serveur vers le client avec un seul appel à `send` et un seul appel à `recv`. Afficher le nombre d'octets envoyés par le serveur, et le nombre d'octets reçus par le client. L'affichage de la valeur du nombre π côté client sera mis en commentaire.

Q2 Quel est le problème ?

Q3 Si le client ne reçoit pas tous les octets envoyés par le serveur, modifier le pour qu'il les reçoive tous, en affichant après chaque appel de `recv` le nombre d'octets reçus. Faire cela sans chercher à placer les 100002 caractères de π dans une seule chaîne.

Les questions suivantes essaieront progressivement de vous faire trouver l'explication au problème constaté. Si pas de problème constaté, aborder quand même les questions suivantes.

2.2. D'où vient le problème ?

Temporiser l'exécution du client avant chaque appel de la fonction `recv`.

Q4 Utiliser les affichages de votre programme et la commande `ss` pour déterminer la relation entre le nombre d'octets envoyés par l'application serveur, le nombre d'octets reçus par l'application cliente, et les nombres d'octets présents dans les tampons internes des sockets.

2.3. Obtenir la taille des buffers internes

Q5 Obtenir la taille du buffer de réception chez le client et la taille du buffer d'envoi chez le serveur. Les afficher.

Ci-après un extrait du code C permettant d'obtenir la taille du tampon interne **d'envoi** associé à une socket `skt` :

```
int sock_buf_size; socklen_t size = sizeof(int);
err = getsockopt(skt, SOL_SOCKET, SO_SNDBUF, &sock_buf_size, &size);
```

L'option utilisée en troisième argument de la fonction est `SO_SNDBUF` pour le buffer d'envoi et `SO_RCVBUF` pour le buffer de réception.

2.4. Modifier la taille des buffers internes

Q6 Modifier la taille du buffer d'envoi chez le serveur, respectivement de réception, chez le client. Les deux seront fixées à 1 octet.

Indication. Utiliser la fonction `setsockopt` déjà vue en TP7 - ci-après son prototype :

```
int setsockopt(int sockfd, int level, int optname,
               const void *optval, socklen_t optlen);
```

Pour la modification de la taille d'un buffer, l'argument `optval` sera l'adresse vers l'entier qui contient la taille à donner au buffer.

Q7 Visualiser les valeurs des tailles affectées aux buffers internes. Correspondent-elles à celles voulues ? Expliquer.
Indication. Fixer les tailles des buffers internes à une autre valeur est possible pour aider à tirer une conclusion.

2.5. Échange du nombre π avec tailles des buffers modifiées

Q8 Modifier éventuellement le client pour qu'il reçoive toujours tous les octets envoyés par le serveur. Utiliser la commande `ss`, et des temporisations avant les appels de `recv`, pour bien comprendre les mécanismes de transferts d'octets mis en jeu.

Q9 Quels sont les effets de la modification de la taille des buffers internes des sockets ?

Q10 Temporiser des deux côtés avant la fermeture des sockets. Visualiser, toujours avec la commande `ss`, l'état des sockets, avant leur fermeture, juste après la fin des applications, et environ 30 secondes plus tard. Tirer des renseignements sur la fermeture d'une connexion.

Q11 Exécuter les applications après avoir mis en commentaire toutes les temporisations.

2.6. Bilan

Q12 De quoi dépend le nombre d'octets lus par un appel de `recv` ?

Q13 Quelles sont les caractéristiques d'une exécution permettant de mettre en évidence l'aspect bloquant de l'appel de `send` ? En conclusion, quand l'appel à `send` rend-il la main ?

Q14 Le nombre d'octets pouvant être envoyés avec un seul appel de `send` est-il limité ?

Q15 Quand un émetteur envoie une quantité importante d'octets, comment faire côté récepteur pour tous les recevoir ?

Q16 Modifier le client pour que, quelles que soient les tailles des buffers internes des sockets, il place toujours les 100002 caractères de π, reçus du serveur, dans une seule chaîne de caractères.